Lógica digital

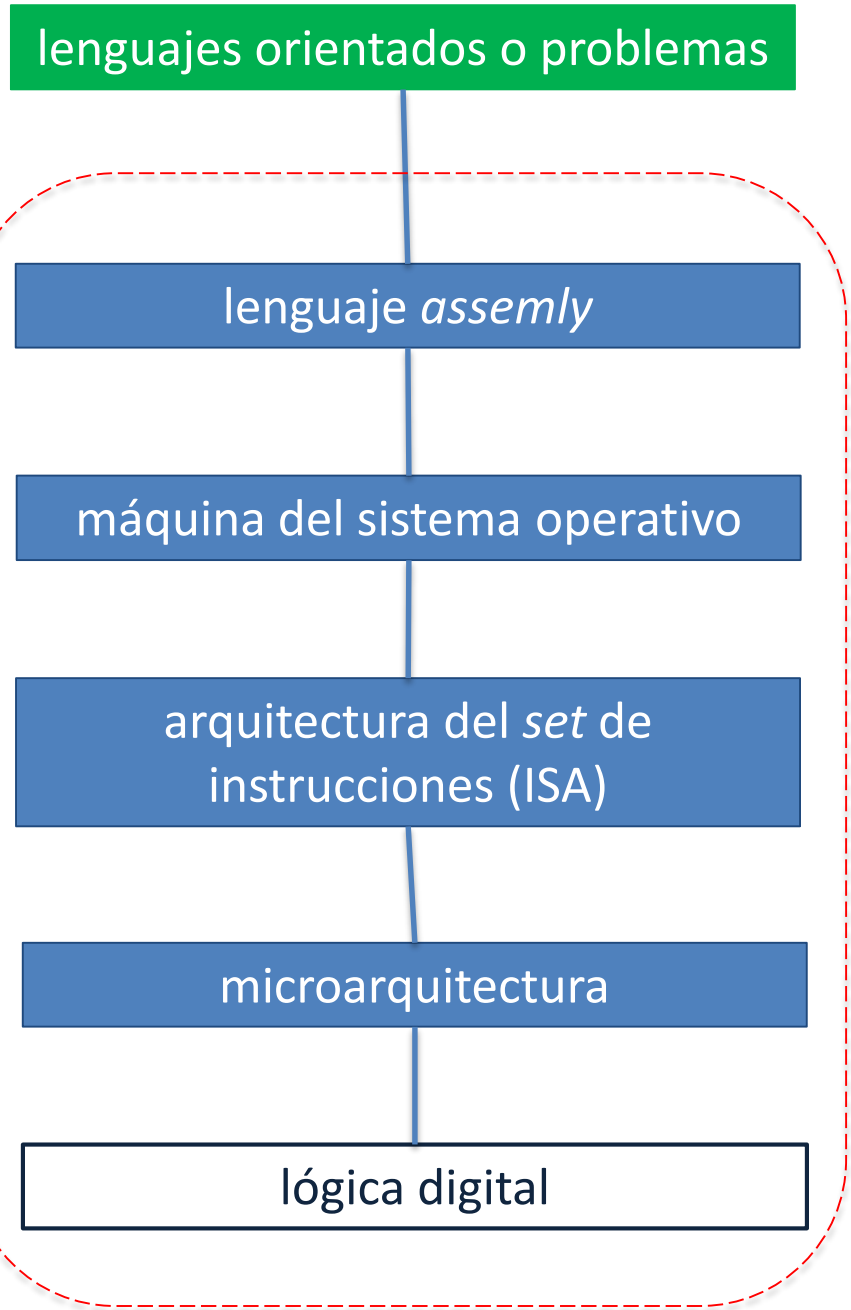
Arquitectura de Computadores – IIC2343

La lógica digital es el verdadero hardware del computador

Conceptualmente, está en la frontera entre la ciencia de la computación y la ingeniería eléctrica

Los elementos básicos a partir de los cuales se construyen todos los computadores digitales son simples:

- los circiutos digitales se construyen a partir de unos pocos elementos primitivos, combinándolos de innumerables formas



5) lenguajes de alto nivel, traducidos por **compiladores**, o interpretados

4) forma simbólica de los lenguajes de más abajo, traducida por el **assembler**

3) instrucciones adicionales, otra organización de memoria, capacidad de ejecución concurrente → interpretado por un programa: el sistema operativo

2) instrucciones ejecutadas por el microprograma o circuitos del hardware

1) 32 registros + **ALU** → **datapath**, **microprogramado** o controlado por hardware

0) compuertas (*gates*), álgebra de Boole, registros, el motor de computación

En un circuito digital hay solo dos valores lógicos:

- una señal entre 0 y 0.5 volts representa el valor binario 0
- una señal entre 1 y 1.5 volts representa el valor binario 1

Las **compuertas** —dispositivos electrónicos pequeños, hechos de transistores— pueden calcular varias funciones a partir de estas señales

… y son la base de hardware sobre la cual se construyen todos los computadores digitales

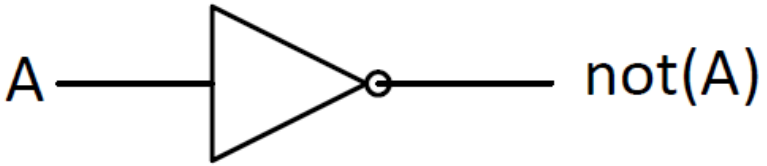
Las próximas diapositivas muestran (los símbolos que se usan para representar) las compuertas NOT (o inversor), AND y OR:

- que junto a las compuertas NAND y NOR son los cinco bloques básicos principales de la lógica digital

… y la función que cada una calcula:

- la tabla debajo del símbolo (y que ya vamos a explicar)

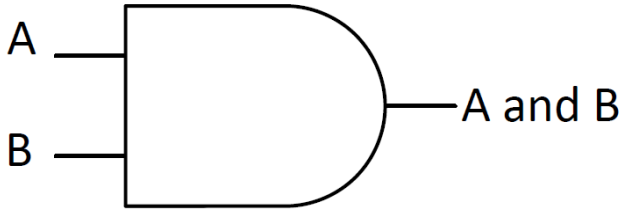
Compuerta NOT:
output = Ã = ¬A



A	not(A)
0	1
1	0

Compuerta AND:

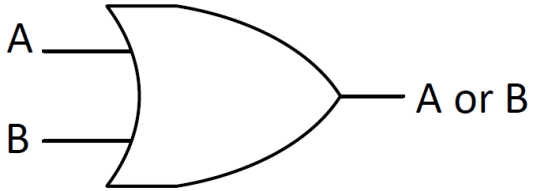
output = AB = A ∧ B



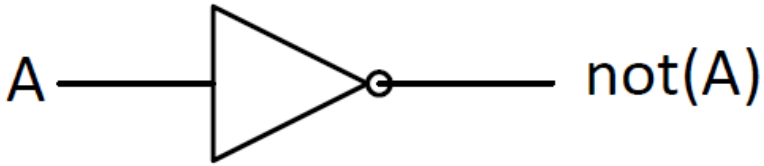
A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

Compuerta OR:

output = A + B = A ∨ B

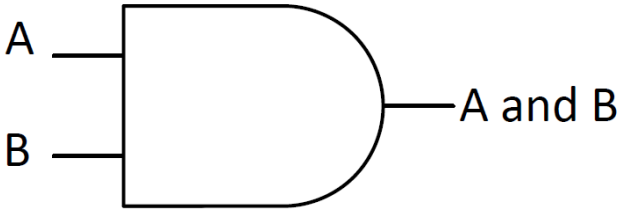


A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1



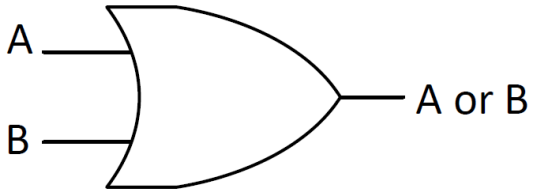
A	not(A)
0	1
1	0

Compuerta NOT: output = Ã = ¬A



A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

Compuerta AND:
output = AB = A ∧ B



A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

Compuerta OR: output = A + B = A ∨ B

Empleamos el **álgebra** (de *switching*) **de Boole** —en que las variables y funciones pueden tomar solo los valores 0 y 1— para describir los circuitos que se pueden construir combinando compuertas

Una función de Boole tiene una o más variables de entrada (*inputs*)

… y produce un resultado (*output*) que depende solo de los valores de las variables de entrada

P.ej., la diapositiva #9 muestra tres funciones de Boole:

- las funciones correspondientes a las compuertas NOT (una variable de entrada: *A*), AND y OR (dos variables de entrada c/u: *A* y *B*)
- … especificadas mediante sendas **tablas de verdad**

P.ej., la siguiente tabla de verdad especifica la función lógica *mayoría* para tres variables:

- vale (el output es) 0 si la mayoría de los inputs son 0
- vale (el output es) 1 si la mayoría de los inputs son 1

A	B	C	output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

¿Cómo podemos describir la función *mayoría* en términos de las variables *A*, *B* y *C* ? :

- especificando cuáles combinaciones de las variables de entrada producen un output de valor 1
- output = …

Otras tres compuertas (y funciones de Boole) comunes, expresadas en función de las compuertas explicadas anteriormente:

NAND —compuerta **completa**, ya que cualquier función de Boole puede ser calculada usando solo compuertas NAND: $\neg(AB)$

NOR —compuerta **completa**, ya que cualquier función de Boole puede ser calculada usando solo compuertas NOR: $\neg(A+B)$

XOR —se puede construir a base de compuertas AND y OR, o a base de compuertas NAND exclusivamente o de compuertas NOR exclusivamente: $A \oplus B = (\neg A \wedge B) \vee (A \wedge \neg B)$

Las compuertas NAND y NOR (ver símbolos y funciones en la pizarra) son las más simples de implementar físicamente —a base de transistores— en la práctica

Para ver la aplicabilidad de estas compuertas y funciones, diseñemos un circuito que sume números (expresados como números binarios)

Las reglas para sumar dos sumandos de un bit son las siguientes:

- 0 + 0 = 0, sin reserva (*carry*)
- 0 + 1 = 1 + 0 = 1, sin reserva
- 1 + 1 = 10, es decir, el dígito correspondiente a la suma es 0 y hay una reserva (*carry*) de 1 que pasa a la próxima posición a la izquierda

Por lo tanto, cuando se suman dos dígitos, *A* y *B*, se producen dos resultados —un sumador de (sumandos de) un bit tiene dos outputs:

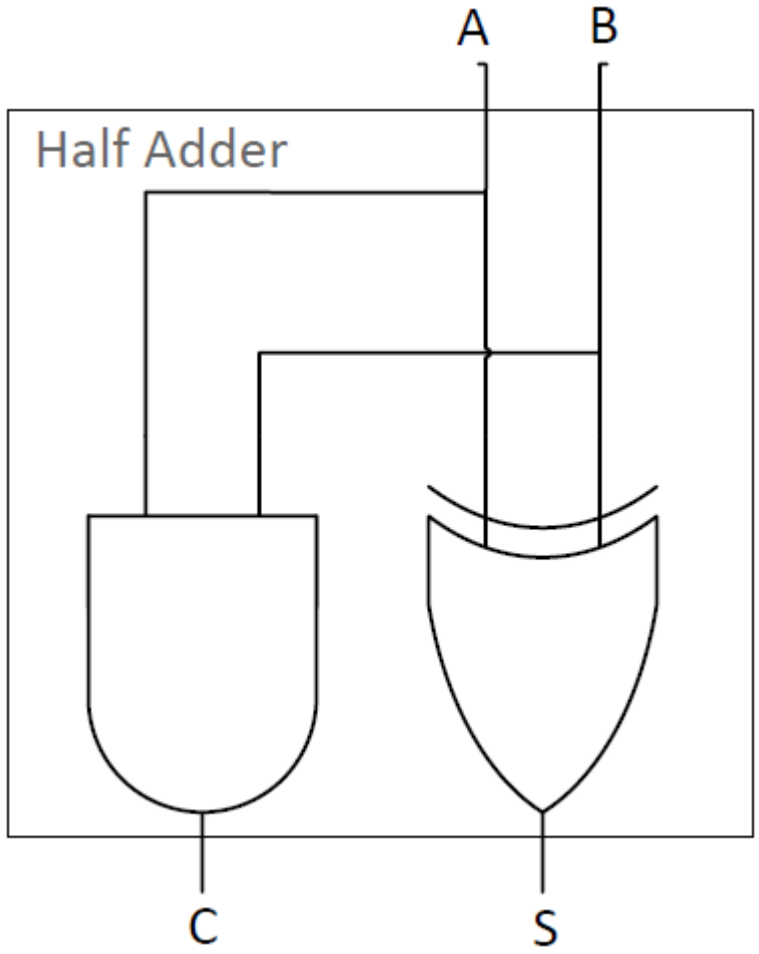
- un dígito, *S*, correspondiente a la suma
- un dígito, *C*, correspondiente a la reserva

He aquí la tabla de verdad y el circuito digital —llamado ***half adder***— correspondiente a un sumador de un bit

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$A \wedge B$

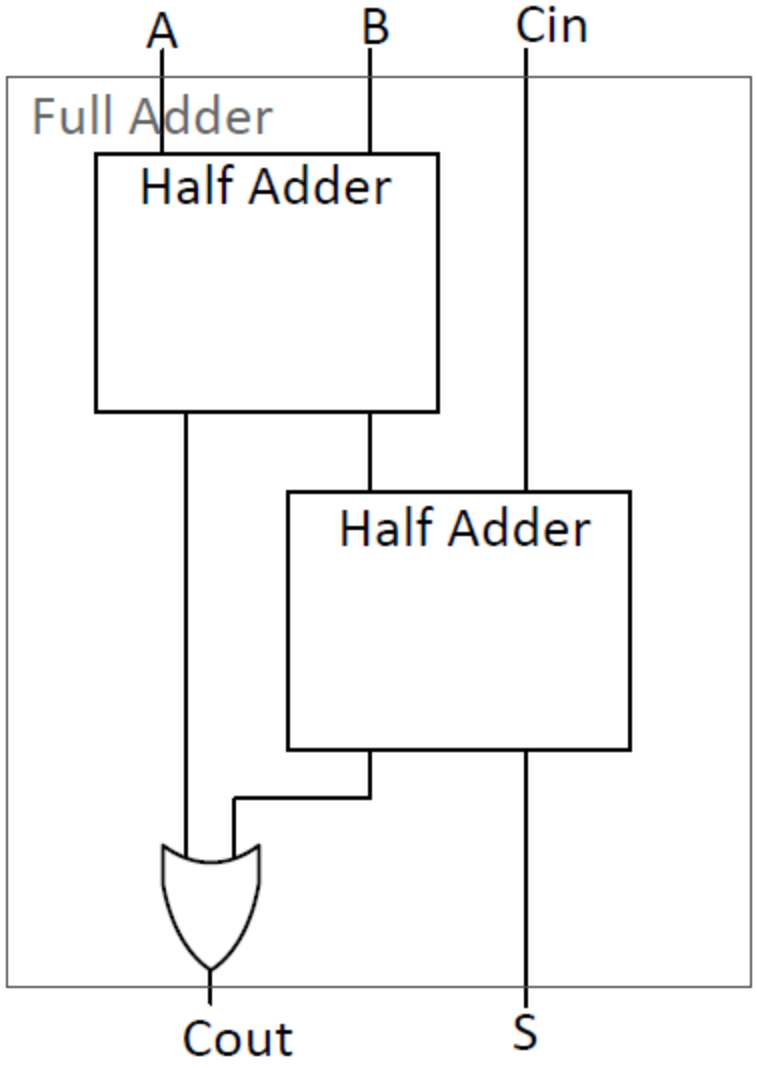
$A \oplus B$



Un ***full adder*** — para sumandos de un bit— es construido a partir de dos *half adders*

Suma tres inputs: *A*, *B* y la reserva, *Cin*, que viene desde la derecha.

Y produce dos outputs: la suma, *S*, y la reserva hacia la izquierda, *Cout*



Por lo tanto, un sumador de sumandos de 4 bits es como se muestra a continuación: